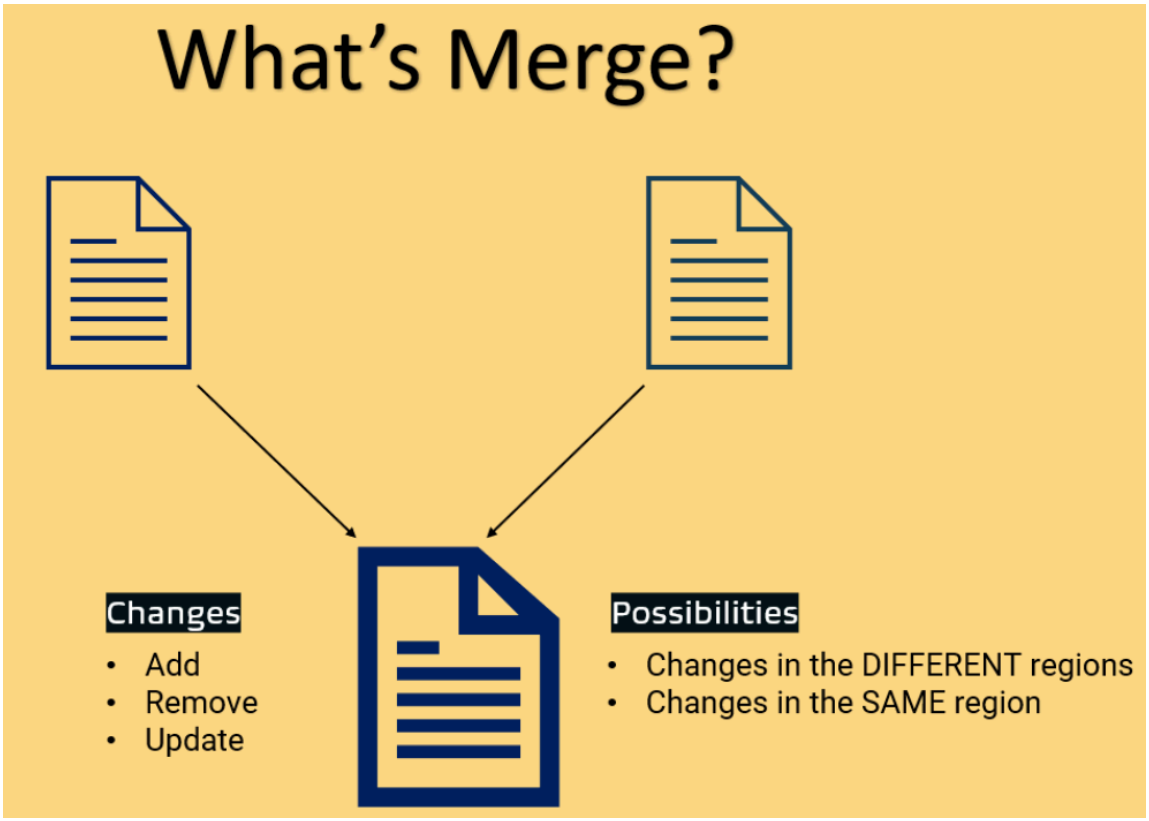
Git Flow Cheat Sheet

Section	Command / Info	Notes
Branch Roles	main	Stable production code
Branch Roles	develop	Integration of new work
Branch Roles	feature/*	Feature development
Branch Roles	release/*	Release prep
Branch Roles	hotfix/*	Urgent production fixes
Start Feature	git checkout develop git pull origin develop git checkout -b feature/your-feature	Branch off from develop
Commit & Push	git add . git commit -m "Add feature XYZ" git push -u origin feature/your-feature	Stage, commit, and push with upstream
Merge Feature	git checkout develop git pull origin develop git merge feature/your-feature git push origin develop	Finish feature via merge to develop
Start Hotfix	git checkout main git pull origin main git checkout -b hotfix/fix-name	Start from main
Finish Hotfix	git add . git commit -m "Hotfix for production issue" git checkout main git merge hotfix/fix-name git push origin main git checkout develop git merge hotfix/fix-name git push origin develop	Merge hotfix into main and develop
Create Release	git checkout develop git checkout -b release/v1.0.0	Create release from develop
Finalize Release	git checkout main git merge release/v1.0.0 git tag -a v1.0.0 -m "Release version 1.0.0" git push origin main --tags git checkout develop git merge release/v1.0.0 git push origin develop	Merge release to main & tag version
Override your local branch	git fetch origin git reset --hard origin/main git clean -fd	Fetch the latest remote info Reset your local branch to match the Remote branch Remove untracked files (e.g., new files not committed)
Utilities	git branch -a	List all branches
Utilities	git branch -d feature/your-feature git push origin --delete feature/your-feature	Delete branches locally and remotely
Utilities	git stash git stash pop	Save and restore uncommitted changes
Best Practices	Use clear branch names like `feature/login`, `hotfix/api-bug`	Helps team collaboration
Best Practices	Always pull latest develop before branching	Avoids conflicts
Best Practices	Use Pull Requests for merging – code review matters	Ensures quality
Delete the .git folder	rm -rf .git rm -r -fo .git	On macOS/Linux On Windows (Git Bash or PowerShell)

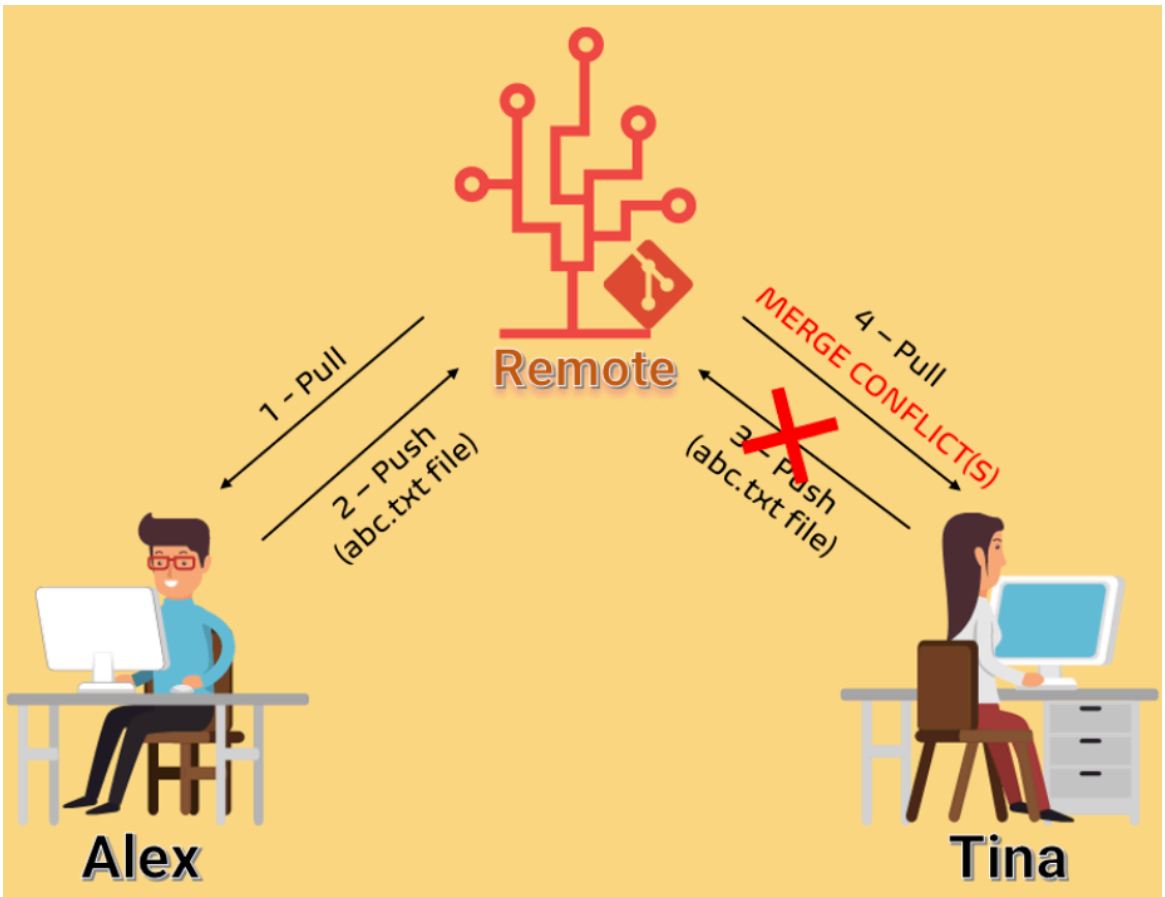
Git Commit Message Standard

Structure: <type>(<scope>): <short description>		
Prefix	**Meaning**	**Example**
Add	Add new features, APIs, tests, or files	feat(user): add registration endpoint
Cut	Remove old features or unused files	chore(payment): remove deprecated payment method
Update	Upgrade logic and improve code for existing features	feat(validation): update validation logic for user input
Enhance	Enhance features, improve UX/UI, or extend functionality	feat(search): enhance search feature with keyword highlight
Improve (synonym)	Optimize to increase accuracy or performance of a feature	perf(checkout): improve checkout flow performance
Fix	Fix bugs, logic errors, or UI issues	fix(login): correct error message on failure
Refactor	If you are only refactoring without changing behavior:	refactor(report): refactor ReportService to use new utility class
	Apply **SOLID** principles	refactor(user): apply Single Responsibility Principle to UserService
	Apply **design patterns**	refactor(order): implement Strategy pattern for discount calculation
	Extract functions to follow **KISS**	refactor(report): split long method into smaller reusable methods
	Remove duplicate code (**DRY**)	refactor(common): extract shared logic into utility class
	Rename variables for clarity (**Naming Convention**)	refactor(product): rename ambiguous variables for clarity
Optimize	If you are updating to improve performance or reduce complexity	perf(image): optimize image resize logic in upload service
Bump	Update versions of packages/modules	chore(deps): bump spring-boot to 3.1.0
Make	Change configuration, Docker, or infrastructure	build(docker): switch to multi-stage Docker build
Start	Initialize a new feature	feat(dark-mode): initial implementation
Stop	Deprecate a feature or API	refactor(auth): remove legacy email login
Style	Adjust formatting, whitespace, Prettier, remove unnecessary comments (**Coding Convention**)	style(codebase): apply Prettier formatting
Document	Update README or code comments	docs(readme): update API usage instructions
Test	Write or update test cases	test(report): add unit tests for summary API
Build	Change the build system (Docker, Webpack, Maven, etc.)	build(maven): add build profile for staging
CI	Configure CI/CD (GitHub Actions, GitLab CI)	ci(gitlab): add deploy stage to pipeline
Chore	Minor tasks that don’t affect business logic	chore(env): rename .env.example
Revert	Revert a previous commit	revert: revert "feat(auth): add login feature"
merge	Use when merging a branch into the main branch	merge(feature/login): integrate login module into develop
chore	Use when you want to keep the commit log clean without generating an automatic changelog	chore: merge hotfix/validate-email into main
release	Use when merging in preparation for release or deployment	release: merge staging into main for deployment

Conflicts in Git



A Horror Story



Example 1: Changes are in the Same Region of the File

The character sequences are like this:

- `<<<<<<<`
- `=======`
- `>>>>>>>`

Everything between <<<<<<< and ======= are your local changes. These changes are not in the remote repository yet.

All the lines between ======= and >>>>>>> are the changes from the remote repository or another branch.

If you need to keep only the line with - Sleep

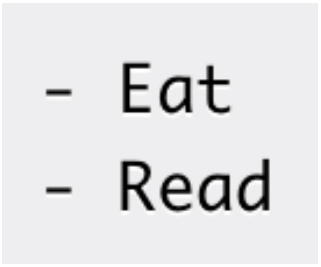
```
- Eat
- Read
- Sleep
```

You can keep the line - Gym and remove the - Sleep changes

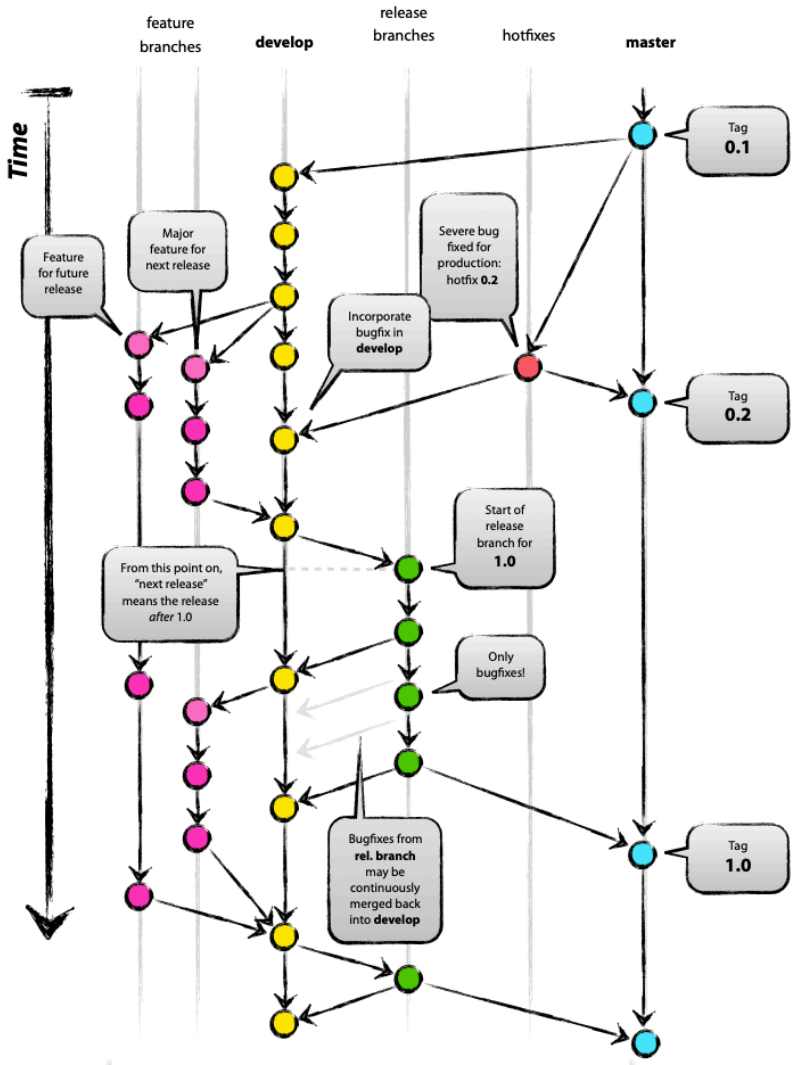
```
- Eat
- Read
- Gym
```

If you need to keep both lines

If you think none of the changes are required, remove them all.



Git Flow



master/main: This is the default branch. It contains the official source code that is released to the community.

hotfixes: This branch is checked out from master/main to quickly fix bugs in master/main. After the fix, it is merged back into master/main.

The reason: we should not modify code directly on master/main, as it is very risky for production.

release: This branch contains completed features currently under testing.

Once testing is finished, it will be merged into master/main.

develop: This branch is checked out from master/main and holds the full commit history of the project.

When a developer completes a feature on a feature branch, it is merged into develop. The leader will review and then merge develop into release for testers to validate.

feature: This branch is checked out from develop. Examples: feature/login, feature/register, etc. After completing a feature, it is merged back into develop.

Hướng dẫn Git cho người mới: git fetch, git diff, git checkout, git pull, git merge, git rebase, git push

1/ Tổng quan

Kết quả khi chạy git fetch

```
kyle@Kys-MacBook-Pro nghianm-reactjs % git fetch
remote: Enumerating objects: 97, done.
remote: Counting objects: 100% (95/95), done.
remote: Compressing objects: 100% (51/51), done.
remote: Total 73 (delta 36), reused 58 (delta 21), pack-reused 0 (from 0)
Unpacking objects: 100% (73/73), 17.93 KiB | 390.00 KiB/s, done.
From https://github.com/minhnghialk/personal-task-management-app
   460a122..5bbc4cd  main       -> origin/main
```

Git phát hiện **có thay đổi** từ nhánh main trên remote.

Git cập nhật branch **origin/main** trên local (nhánh theo dõi remote).

Nhánh main trên local của bạn vẫn **chưa bị thay đổi**.

=> Muốn update về local:

1. git pull origin main

Kết quả khi git fetch với nhiều nhánh

```
kyle@Kys-MacBook-Pro nghianm-reactjs % git fetch
remote: Enumerating objects: 252, done.
remote: Counting objects: 100% (252/252), done.
remote: Compressing objects: 100% (117/117), done.
remote: Total 233 (delta 140), reused 208 (delta 115), pack-reused 0 (from 0)
Receiving objects: 100% (233/233), 117.36 KiB | 234.00 KiB/s, done.
Resolving deltas: 100% (140/140), completed with 15 local objects.
From https://github.com/minhnghialk/personal-task-management-app
 + 0e37284...bb6fc97 feature/dashboard-sidebar -> origin/feature/dashboard-sidebar  (forced update)
 + 72a246e...15fd5d0 feature/create-task       -> origin/feature/create-task  (forced update)
   e8a0552..70ce95c  feature/login             -> origin/feature/login
```

Git cập nhật 3 nhánh:

- feature/dashboard-sidebar bị force update (ai đó dùng git push --force)
- feature/create-task bị force update
- feature/login cập nhật bình thường

2/ git fetch là gì?

- Là lệnh để lấy về **toàn bộ commit mới** từ remote repo.
- KHÔNG tự động gộp vào local branch.
- Giúp ta kiểm tra xem remote đang thay đổi như thế nào.

Lệnh:

1. **git fetch origin**

Sau khi fetch:

1. **git diff main origin/main**

=> So sánh sự khác nhau giữa local vs remote.

3/ git pull là gì?

Là **kết hợp fetch + merge**

Sự dụng: khi bạn muốn lấy code mới về và đảm bảo local mình đặt theo remote.

Lệnh:

1. **git pull origin main**

Tương đương: git fetch origin + git merge origin/main

4/ git merge là gì?

Dùng để **gộp nhánh** khác vào nhánh hiện tại

VD:

1. **git checkout feature/login**
2. **git merge develop**

=> Gộp code từ develop vào feature/login

5/ git rebase là gì?

Ví dụ:

Bạn đang làm việc ở nhánh feature/login, có lịch sử:

develop: A --- B --- C

feature/login: A --- D --- E

Team bạn đã cập nhật develop thành:

develop: A --- B --- C --- F --- G

Nếu bạn muốn cập nhật develop mới nhất mà vẫn giữ commit D → E, bạn dùng:

1. **git checkout feature/login**
2. **git fetch origin**
3. **git rebase origin/develop**

→ Kết quả: feature/login: A --- B --- C --- F --- G --- D' --- E'

(D' và E' là bản sao mới của D và E, sau khi "rebase" lên nền G)

6/ git push —force

Khi bạn đã **thay đổi lịch sử commit** (bằng git rebase hoặc git commit --amend) → local của bạn **khác lịch sử remote**.

Lúc này Git **không cho push bình thường** vì sẽ gây mất commit.

Nếu bạn dùng git push --force, Git sẽ ép remote phải giống hệt local của bạn, **xóa bỏ commit từ người khác** nếu có.

👉 Điều này nguy hiểm khi làm việc nhóm, vì bạn có thể vô tình **đè mất code** đồng đội vừa push.

7/ git push --force-with-lease là gì

Nó cũng ép buộc như --force, nhưng **có thêm một lớp an toàn**:

- Trước khi ghi đè, Git sẽ kiểm tra xem remote branch có thay đổi gì **ngoài những gì bạn đã fetch về** hay không.
- Nếu remote có commit mới từ đồng đội mà bạn chưa fetch, **Git sẽ từ chối push**.

👉 Điều này giúp tránh việc **ghi đè mất commit của người khác**.

Ví dụ: Giả sử nhánh feature/login

Tình huống 1: Bạn làm việc một mình

- Bạn commit A → B → C và push.
- Sau đó, bạn rebase thành A → D → E.
- Lúc này local khác remote → bạn push:

1. **git push --force-with-lease**

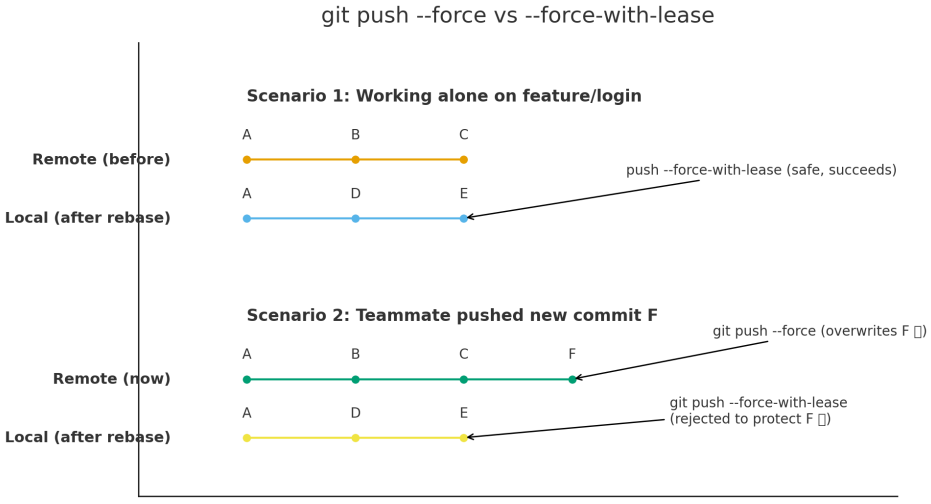
👉 Thành công vì chỉ bạn thay đổi branch đó.

Tình huống 2: Làm việc nhóm

- Remote hiện có: A → B → C
- Bạn rebase thành: A → D → E
- Đồng đội push thêm commit F (bạn chưa biết).

👉 Nếu bạn git push **--force** → **commit F của đồng đội sẽ bị mất** ❌.

👉 Nếu bạn git push **--force-with-lease** → Git từ chối, báo lỗi ❌ → giúp bạn nhận ra cần git fetch trước.



Khi nào nên dùng?

- **Làm việc một mình trên nhánh cá nhân** → có thể dùng --force hoặc —force-with-lease.
- **Làm việc nhóm trên nhánh chung (develop, feature chung, main)** → luôn dùng: git push --force-with-lease

Fast-forward là gì? và chúng ta làm gì khi không thể fast-forward?

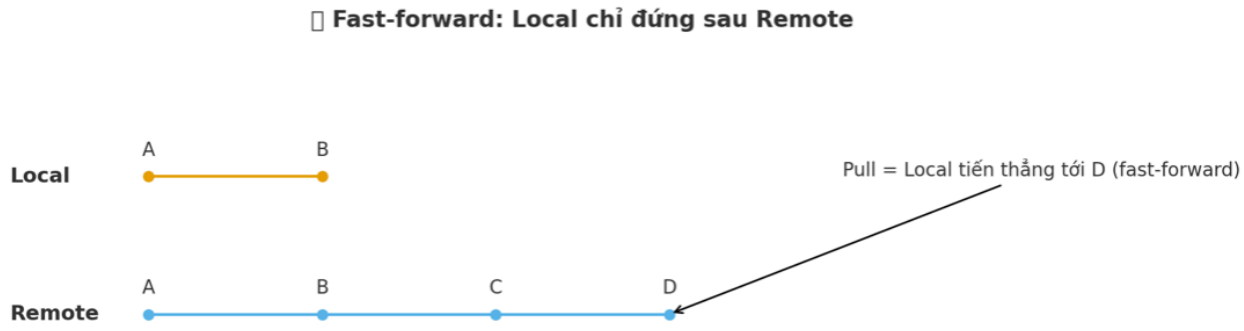
Tình huống: Khi bạn git pull và gặp “fatal: Not possible to **fast-forward**, aborting.”

Thống nhất cách hiểu: Khi git pull, Git sẽ cập nhật **local** branch để giống **remote**.

Trường hợp 1: Local **chưa có commit riêng nào**

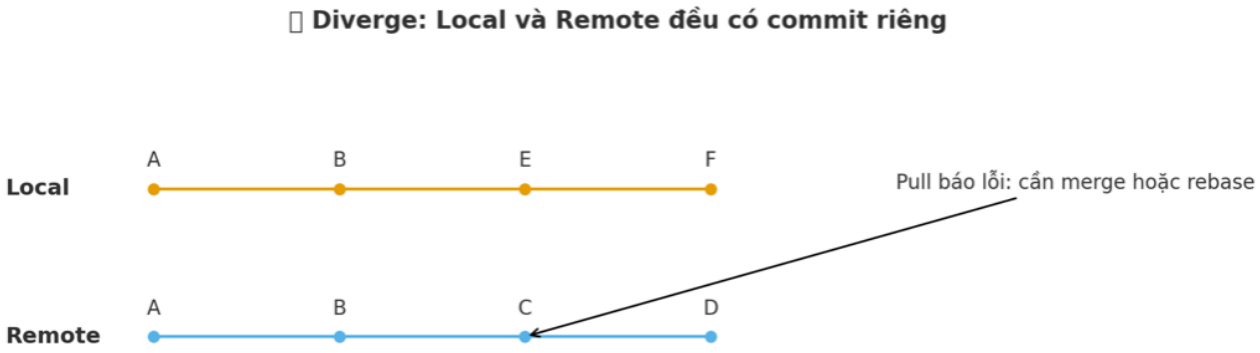
- Ví dụ:
  - Local : A → B
  - Remote: A → B → **C** → **D**

👉 git pull: thì local chỉ cần “tiến tới D” là xong (fast-forward). Vì local chỉ thiếu commit C, D so với remote → git pull chỉ “tiến thẳng” tới D, không cần merge.



Trường hợp 2: Cả hai bên đều **có commit riêng**

- Ví dụ:
  - Remote: A → B → **C** → **D** (Remote có thêm C, D)
  - Local : A → B → **E** → **F** (Local có thêm E, F)



- Khi bạn git pull: “**fatal: Not possible to fast-forward, aborting.”** Git không thể chỉ “nhảy thẳng” từ B lên D, vì sẽ mất E, F.
- Nghĩa là: **“Không thể fast-forward, vì cả hai bên đều có commit riêng.”** Lúc này Git không thể fast-forward vì sẽ bỏ sót commit của bạn hoặc commit của remote.
- Bạn phải **merge, rebase hoặc reset (Chi tiết xem bên dưới)**

Chúng ta làm gì khi không thể fast-forward (diverge)?

Bạn phải chọn 1 trong 3 cách sau:

1. **Merge:** Giữ cả hai nhánh và thêm commit merge:

   A → B → C → D

   \ \

   E → F → (merge)

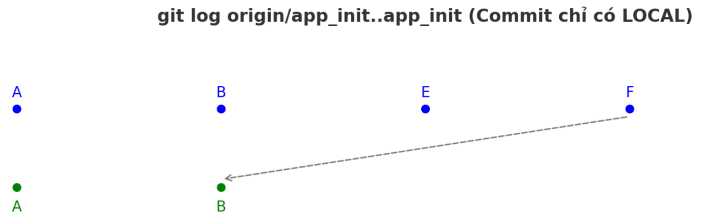
2. **Rebase:** Đặt commit local (E, F) "lên trên" commit mới từ remote:

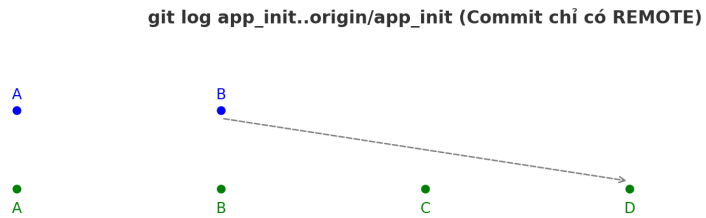
   A → B → C → D → E' → F'

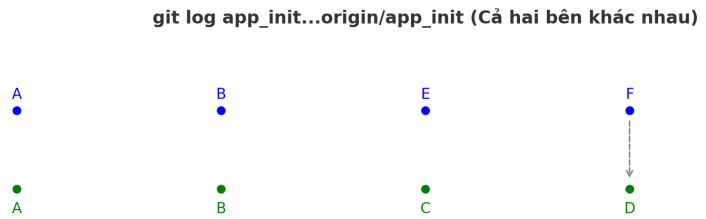
3. **Reset (override):** Nếu bạn muốn bỏ local và chỉ giữ remote:

   A → B → C → D

Lời khuyên: Khi Git báo "**Not possible to fast-forward (diverge)**", bạn nên **so sánh lịch sử commit giữa local và remote** để biết nhánh đã khác nhau chỗ nào.







Cách đọc:

1. **git log origin/app_init..app_init**
   → Commit màu xanh dương (E, F) chỉ có ở local.
2. **git log app_init..origin/app_init**
   → Commit màu xanh lá (C, D) chỉ có ở remote.
3. **git log app_init...origin/app_init**
   → Hiện cả hai phía: Local có (E, F), Remote có (C, D).